Towards Time-Triggered component-based system models

Hela GUESMI, Belgacem BEN HEDIA — CEA-LIST, PC 172, 91191 Gif-sur-Yvette, France, firstname.lastname@cea.fr

Simon BLIUDZE — EPFL IC IIF RiSD, Station 14, CH-1015 Lausanne, Switzerland, simon.bliudze@epfl.ch

Saddek BENSALEM, Jacques COMBAZ — Verimag, 38610 Gieres, France, firstname.lastname@imag.fr

***Abstract*—In this paper, we propose a methodology for producing correct-by-construction Time-Triggered (TT) physical model by starting from a high-level model of the application software in BIP. BIP (Behavior, Interaction, Priority) is a component based framework with formal semantics that rely on multi-party interactions for synchronizing components. Commonly in TT implementations, processes interact with each other through a communication medium. Our methodology transforms, depending on a user-defined task mapping, high-level BIP models where communication between components is strongly synchronized, into TT physical model that integrates a communication medium. Thus only inter-task communications and components participating in such interactions are concerned by the transformation process. The transformation consists of: (1) breaking atomicity of actions in components by replacing strong synchronizations with asynchronous send/receive interactions, (2) inserting communication media that coordinate execution of inter-task interactions according to a user-defined task mapping, (3) extending the model with an algorithm for handling conflicts between different communication media and (4) instantiating task components and adding local priority rules for handling conflicts between inter-task and intra-task interactions. We also prove the correctness of our transformation, which preserves safety properties.**

Keywords–Anything; Time-Triggered paradigm; correct-by-construction; model transformation; BIP.

I. INTRODUCTION

A Time-Triggered (TT) system initiates all system activities -task activation, message transmission, and message detection- at predetermined points in time. Ideally, in a time-triggered operating system there is only one interrupt signal: the ticks generated by the local periodic clock. These statically defined activation instants enforce regularity and make TT systems more predictable than Event-Triggered (ET) systems. This approach is well-suited for hard real-time systems.

Kopetz [1] [2] presents an approach for real-time system design based on the TT paradigm which comprises three essential elements:

The global notion of time: It must be established by a periodic clock synchronization in order to enable a TT communication and computation,

The temporal control structure of each task: In a sequence of computational or communication processes (called *tasks*), the start of a task is triggered by the progression of the global time, independently from the involved data of the task. The worst-case execution time and thus the worst-case termination instant are also assumed to be known a priori. These statically predefined start and worst-case termination instants, define the temporal control structure of the task,

TT communication system: To isolate subsystems from each other in a TT architecture, a special interface called the temporal firewall has been designed. It consists in shared memory element for unidirectional exchange of information between sender/receiver tasks components. It's the responsibility of the TT communication system [3] [1] to transport, with access to the global time, the information from the sender firewall to the receiver firewall. The instants at which information is delivered or received are a priori defined and known to all nodes. Furthermore, the TT communication service/protocol avoids interference between concurrent read and write operations on the memory elements.

Analysis and design of hard real-time systems often starts with developing a high-level model of the system. Building models allows designers to abstract away implementation details and validate the model regarding a set of intended requirements through different techniques such as formal verification, simulation, and testing. However, deriving a correct TT implementation from a high-level model is always challenging, since adding TT implementation details involves many subtleties that can potentially introduce errors to the resulting system. Furthermore, in the context of hard real-time systems (and time-triggered paradigm), services offered by target operating systems should be taken into account in the derived implementation.
Thus it is highly advantageous if designers can somehow derive a model with implementation details in a systematic and correct way from high-level models. We call such a model physical model. It can be automatically translated to the programming language specific to the target TT platform.

In this paper, we present a method for transforming high-level models in BIP [4] into TT physical model that integrates the three TT-paradigm properties mentioned above. The BIP framework is used for constructing systems by superposition of three layers: Behaviour, Interaction, and Priority. The Behaviour layer consists of a set of components represented by automata or Petri nets extended by data and functions given in C++. The interaction layer describes possible interactions between atomic components. Interactions are sets of ports allowing synchronizations between components. They are defined and graphically represented by connectors. The execution of interactions may involve transfer of data between the participating components. The third layer includes priorities between interactions using mechanisms for conflict resolution. In this work, we consider Real-Time BIP (RT-BIP) framework [5] [6] where behaviour of components is represented by a timed automaton [7].

We believe that the first two properties of the TT paradigm

can be obtained straightforwardly from RT-BIP, and that the main difficulty lies in representing tasks and inter-task communication system in the physical model. To achieve this goal, we introduced in this paper TT-BIP model, which combines elements of both RT-BIP, Send/Receive [8] (SR-BIP) model and TT paradigm. It introduces additional structure in order to model the TT communication system explicitly. The TT communication system is modelled by introducing dedicated atomic components. Depending on a user-defined task mapping, components in the system executing the same task are grouped into composite components called *tasks*. The latter can interact only through the atomic components modelling the TT communication system. The TT-BIP model draws on some of the ideas behind SR-BIP – the model used for distributed implementation of untimed BIP systems [9] and RT-BIP systems [10]. We also define transformation rules for deriving a TT-BIP model from a high-level RT-BIP model and a task mapping. We show that this transformation preserves trace inclusion (i.e safety properties).

The rest of this paper is structured as follows. In Section II, we present the basic concepts related to our work; RT-BIP and SR-BIP models. Section III formalizes the TT-BIP architecture and describes the transformation. Experimental results are presented in Section IV. Section V analyses related work.And finally, we make concluding remarks in VI. For reason of space, all correctness proofs appear in the appendix.

II. Background concept: BIP Framework

In this section, we first provide the definition and the semantics of RT-BIP. RT-BIP is a component framework for constructing systems by superposing three layers of modelling: Behaviour, Interaction, and priority. And then, we describe the SR-BIP model. Before giving the definition and semantics of an RT-BIP component, we first fix some notations.

A. Preliminary notations

1) Valuation function: Given a variable x, the domain of x is the set $D(x)$ of all values possibly taken by x. Given a set of variables X, We denote by $v(x)$ the corresponding element of $x \in D(x)$. A valuation of X is a function $v : X \to \bigcup_{x \in X} D(x)$ associating with each variable x its value $v(x)$. Given a subset of variables $X' \subseteq X$ and a variable value $a \in D(x)$, we denote by $v\,[X' \leftarrow a]$ the valuation defined by:

$$v[X' \leftarrow a](x) = \begin{cases} a & \text{if } x \in X' \\ v(x) & otherwise. \end{cases} \quad (1)$$

We denote by $\mathcal{V}(X) = \prod_{x \in X} D(X)$ the set of all possible valuations of the variables in X.

2) Guards: Guards are boolean expressions used to specify when actions of a system are enabled. A guard g is a predicate on a set of variables X. Given valuation $v \in \mathcal{V}(X)$, we denote by $g(v) \in \{\textit{False}, \textit{True}\}$ the evaluation of g for v.

3) Clocks: We assume that time progress is measured by clocks which are integer or real-valued variables increasing synchronously. Each clock can be reset (i.e. set to 0) independently of other clocks.
We denote by $\mathbb{R}_+$ the set of non-negative reals, and by $\mathbb{N}$ the set of non-negative integers. Given a set of clocks C, let $\mathcal{V}(C)$ be the set of all clock valuation functions $v_c : C \to \mathbb{R}_+$.
Given δ, such that $\delta \in \mathbb{R}_+$, for all $c \in C$, we use $c+\delta$ as usual notation for the valuation defined by $(v_c + \delta)(c) = v_c(c) + \delta$.

4) Timing constraints: Timing constraints are guards over the set of clocks C. They are used to specify when actions of a system are enabled regarding system clocks. The basic building blocks for timing constraints are comparisons; given a set of clocks C, $c \in C$ and $a \in \mathbb{R}_+$, comparison between the valuation of c and a can be presented as $c \sim a$ where $\sim \in \{\leq, <, =, >, \geq\}$. Constraints are built using the following grammar:

$$tc := \textit{True} \mid \textit{False} \mid c \sim a \mid tc \wedge tc \mid tc \vee tc \mid \neg tc$$

Notice that any guard tc can be written as:

$$tc := \bigwedge_{c \in C} l_c \leq c \leq u_c, \text{ where } l_c, u_c \in \mathbb{R}_+ \forall c \in C \quad (2)$$

We denote by $\mathcal{TC}(C)$, the set of clock constraints defined over clocks of C.

5) Time progress conditions: Time progress conditions are used to specify whether time can progress at a given state of the system. They correspond to a special case of timing constraint where $\sim$ is restricted to $\{\leq\}$ and operators $\neg$ and $\vee$ are disallowed. Formally, time progress conditions are defined by the following grammar:

$$tpc := \textit{True} \mid \textit{False} \mid c \sim a \mid tc \wedge tc, \text{ where } c \in C \text{ and } a \in \mathbb{R}_+$$

Note that any time progress condition tpc can be written as:

$$tpc = \bigwedge_{c \in C} c \leq u_c, \text{ where } u_c \in \mathbb{R}_+ \cup \{+\infty\} \quad (3)$$

B. Basic semantic model of RT-BIP

In RT-BIP, systems are built by composing *atomic components* with *interactions* defined using *connectors*.

A *component* in RT-BIP is essentially a timed automaton [11] labelled by ports that represent the component's interface for communication with other components. Let $\mathcal{P}$ be a set of ports. We assume that every port $p \in \mathcal{P}$ has an associated data variable x_p. This variable is used to exchange data with other components, when interactions take place.

***Definition** 1:* (Component). A component B is a tuple $B = (L, P, X, C, T, \{tpc_l\}_{l \in L})$ where: L is a finite set of control locations, $P \subseteq \mathcal{P}$ is a finite set of ports, called the interface of B, X is a set of local variables. We denote the set of variables associated to ports by $X_P \subseteq X$ and the set of the rest of local variables by X_B, such that $X = X_B \cup X_P$, C is a finite set of clocks, T is a set of labelled transitions. A transition $\tau \in T$ from a control location l to l' is a tuple $\tau = (l, p, g_\tau, f_\tau, R, l')$ where :

- p is a port.
- $g_\tau = g^X \wedge tc$ is a boolean guard, which is a conjunction of a predicate g^X on local variables X and a timing constraint tc over C. We say that a transition τ is enabled for the valuation $v \in \mathcal{V}(X)$, when its guard g_τ evaluates to *True*.
- f_τ is a function that updates the set of variables on X
- R is the subset of clocks $R \subseteq C$, that are reset by the transition τ.

For each place $l \in L$, tpc_l is a time progress condition.

***Definition** 2:* (Semantics of a component). The semantics of a component $B = (L, P, X, C, T, \{tpc_l\}_{l \in L})$ is defined as a labelled transition system $S_B = (Q_B, P_B, \xrightarrow[B]{})$, where: $Q_B =$

$L \times \mathcal{V}(C) \times \mathcal{V}(X)$ is the set of states, $P_B = P \cup \mathbb{R}_{\geqslant 0}$ is the set of labels: ports or time delays, $\xrightarrow[B]{} \subset Q_B \times P_B \times Q_B$ is the set of labelled transitions defined as follows. Let (l, v_c, v_x) and (l', v'_c, v'_x) be two states, $p \in P$, and $\delta \in \mathbb{R}_{\geqslant 0}$ be a delay.

- **Jump transitions**: We have $(l, v_c, v_x) \xrightarrow[B]{p} (l', v'_c, v'_x)$ iff there exists a transition $\tau = (l, p, g_\tau, f_\tau, R, l')$ enabled for (v_c, v_x) and $v'_x = f_\tau(v_x)$, and for all $c \in r$, $v'_c(c) = 0$.
- **Delay transitions**: We have $(l, v_c, v_x) \xrightarrow[B]{\delta} (l, v_c + \delta, v_x)$ iff $\forall \delta' \in [0, \delta]$, $tpc_l(v_c + \delta')$ evaluates to *True*.

A component B can execute a transition $\tau = (l, p, g_\tau, f_\tau, R, l')$ from a state (l, v_c, v_x) if its timing constraint is met by the valuation v_c. The execution of τ corresponds to moving from control location l to l', updating variables and resetting clocks of R. From state (l, v_c, v_x), B can also wait for $\delta > 0$ time units if the time progress condition tpc_l stays *True*. Waiting for δ time units increases all the clock values by δ. Notice that the execution of transitions is instantaneous and time elapses only on states.

The interaction model is specified by a set of interactions $\gamma \subseteq 2^P$. Interactions of γ can be enabled or disabled.

***Definition** 3:* (Interaction). Let $\{B_i\}_{i=1}^n$ be a set of components as above. An interaction α between components $\{B_i\}_{i=1}^n$ is a quadruple (a, X_a, G_a, F_a), where: $a \subseteq P$ contains at most one port of every component, that is, $\mid a \cap P_i \mid \leqslant 1$, for all $i \in [1, n]$. $X_a = \cup_{p \in a} X_p$ is the set of variables available to an interaction α. G_a is the set of boolean guards associated to α. F_a is the set of the update functions associated to α and defined over X_a.

In the remainder of the paper, we may denote the interaction (a, X_a, G_a, F_a) by its set of ports a. An interaction a is enabled for a valuation v_a of X_a if and only if, for all $i \in [1, n]$, the port in $a \cap P_i$ is enabled in B_i and $G_a(v_a) =$ *True*. That is, an interaction is enabled if each port that is participating in this interaction is enabled and the guard evaluates to *True*.

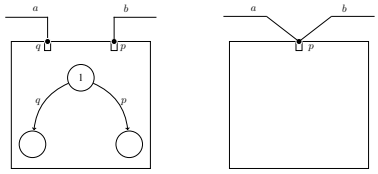
We denote by $comp(a)$ the set of components that have ports participating in a. $comp(a)$ is formally defined as:

$$comp(a) = \{B_i | i \in [1, n], P_i \cap a \neq \emptyset\} \tag{4}$$

Two interactions are conflicting at a given state of the system if both are enabled, but it is not possible to execute both from that state (i.e. the execution of one of them disables the other). In fact the enabledness of interactions only indirectly depends on the current state, through the enabledness of the participating ports. In systems without priorities, two interactions a and b may conflict only if they involve a shared component. In Figure 1a, the conflict comes from the fact that a and b involve two ports p and q of the same component labelling two transitions enabled from the same location. When reaching location, the component can execute either transition labeled by p or the one labeled by q but not both. This implies that when a and b are enabled, only one of them should execute. Figure 1b depicts a special case of conflict where interactions a and b share a common port p. Update functions of a and b may update variables exported by port p. This implies that when a and b are enabled, only one of them should execute.

***Definition** 4:* (Conflicting interactions).

Let $\gamma(B_1, ..., B_n)$ be a BIP model. We say that two interactions a and b of γ are conflicting, iff, there exists an atomic component $B_i \in comp(a) \cap comp(b)$ that has two transitions $\tau_a = (l, p, l'_1)$ and $\tau_b = (l, q, l'_2)$ from the same control location l such that $p \in a$ and $q \in b$. We denoted the conflict between a and b by $a \# b$. If a and b are not conflicting we say that they are independent. The system is conflict-free if all interactions are pairwise-independent.



(a) (b)

Figure 1. Conflicting interactions

Priorities are defined in order to reduce non-determinism in the system, that is, they are used to filter interactions among the enabled ones.

***Definition** 5:* (Priority in BIP).

Given a set γ of interactions defined over a set of components $\{B_1, ..., B_n\}$, we define a priority as a relation $\pi \subseteq \mathbb{B}^Q \times \gamma \times \gamma$, where $\mathbb{B}$ is the set of booleans and Q the set of states, such that for all $(C, a, a') \in \pi$, C depends only on data variables that are associated with ports of interactions a or a' and $\forall q \in Q$, $\pi_q = \{(a, a') \in \gamma \times \gamma \mid C(q) \wedge (C, a, a') \in \pi\}$ is a partial order. We say that a has less priority than a' whenever the predicate C holds.

The predicate C depends on the data variables exported by the participants in some interactions, allowing the corresponding priority rule to be dynamically enabled. A static priority is expressed by having $C =$ *True* for all $(C, a, a') \in \pi$.

A composite component is built from a set of n components $\{B_i = (L_i, P_i, X_i, C_i, T_i, \{tpc_l\}_{l \in L_i})\}_{i=1}^n$ such that their respective sets of places, ports, clocks, and discrete variables are pairwise disjoint; i.e., for any two $i \neq j$ from $[1, n]$, we have $L_i \cap L_j = \emptyset$, $P_i \cap P_j = \emptyset$, $C_i \cap C_j = \emptyset$, and $X_i \cap X_j = \emptyset$.

***Definition** 6:* (Composite Component with interactions). Let γ be a set of interactions. We denote by $B \stackrel{def}{=} \gamma(B_1, ..., B_n)$ the composite component obtained by applying γ to the set of components $\{B_i\}_{i=1}^n$. It is defined by the component $B = (L, P, X, C, T, \{tpc_l\}_{l \in L})$ as follows: $L = L_1 \times ... \times L_n$ is the set of locations, $P = \bigcup_{i=1}^n P_i$ is the set of ports, $X = \bigcup_{i=1}^n X_i$ is the set of variables, $C = \bigcup_{i=1}^n C_i$ is the set of clocks, Let $\alpha = (a, X_a, G_a, F_a) \in \gamma$ be an interaction. We denote $I_a = i \in [1, n] | B_i \in comp(a)$. A transition $\tau = (l, a, g_\tau, f_\tau, R, l')$ from $l = (l_1, ..., l_n)$ to $l' = (l'_1, ..., l'_n)$ ($l = l'$ if $i \notin I_a$) is in T if its projection $\tau_i = (l_i, p_i, g_{\tau_i}, f_{\tau_i}, R_i, l'_i)$ is a transition of B_i for all $i \in I_a$, where $g_{\tau i}$ and $f_{\tau i}$ are such that :

- $g_\tau = G_a \wedge \bigwedge_{i \in I_a} g_{\tau_i}$,
- $f_\tau = f_{\tau_1} \circ ... \circ f_{\tau_n} \circ F_a$ where f_{τ_i} is the identity function, for $i \notin I_a$ (Notice that functions f_{τ_i} modify disjoint sets of variables, hence can be composed in any order),
- $R = \bigcup_{i=I_a} R_i$.

For a control location $l = (l_1, ..., l_n) \in L$, the time progress condition is $tpc_l = \bigwedge_{i \in [1,n]} tpc_{l_i}$.

***Definition** 7:* (Composite Component with priorities). Let $B^\gamma = \gamma(B_1, ..., B_n)$ be the composite component obtained by applying γ to the set of components $\{B_i\}_{i=1}^n$. And let L be their set of locations and π the set of priority rules π_l for $l \in L$. We denote by $B^{\pi\gamma} = \pi\gamma(B_1, ..., B_n)$ the composite component obtained by applying priorities to the set of components B^γ: Given two interactions $a, a' \in \gamma$, with corresponding transitions $\tau^\gamma = (l, a, g_{\tau^\gamma}, f_{\tau^\gamma}, R_a, l')$ and $\tau'^\gamma = (l, a', g_{\tau'^\gamma}, f_{\tau'^\gamma}, r_{a'}, l'')$ in B^γ, such that $\exists (C, a, a') \in \pi$ (i.e. a has less priority than a' in the current location l), the corresponding transitions in $B^{\pi\gamma}$ are:

- $\tau^{\pi\gamma} = (l, a, g_{\tau^{\pi\gamma}}, f_{\tau^\gamma}, R_a, l')$, with $g_{\tau^{\pi\gamma}} = g_{\tau^\gamma} \wedge \neg g_{\tau'^\gamma} \wedge C$,
- and $\tau'^{\pi\gamma} = (l, a', g_{\tau'^{\pi\gamma}}, f_{\tau'^\gamma}, R_{a'}, l'')$, with $g_{\tau'^{\pi\gamma}} = g_{\tau'^\gamma}$.

The execution of interactions, taking into account priority rules and execution of local code of components are orchestrated by a sequential *scheduler*. Conflicts that are not resolved by priority rules are resolved by this scheduler; it randomly chooses one of simultaneously enabled interactions.

Components that are not composite, i.e. specified directly as LTS in the model, are called in the remainder of the article atomic components.

C. SR-BIP model

The SR-BIP models are designed, for untimed [8] [12] and timed [10] BIP, to automatically derive distributed implementations. They aim to be implementable using basic message-passing primitives. The execution of a distributed process is a sequence of actions that are either message emission, message reception or internal computation. Consequently the SR-BIP model includes three types of ports: send-ports, receive-ports and unary-ports. Unary-ports correspond to internal computation. They can only appear in unary interactions, that is interactions involving only one component. Send and receive ports appear only in message-passing interactions (called send/receive interactions). Such an interaction has no guard, and the update function copies variables exported by the send-port to variables exported by the receive-port. In a canonic message-passing environment, each send action has a well-defined recipient. Therefore, it is required in SR-BIP models that each send-port participates in exactly one send/receive interaction. The latter ensures that for each send-port there is a unique corresponding receive-port.

An SR-BIP model is an RT-BIP model which contains components glued by send/receive interactions. A send/receive interaction is composed of one send-port and one or more receive-ports depending on the data transfer direction.

Since concurrency and distribution introduced to RT-BIP model can not be handled by sequential single scheduler, SR-BIP model handles interactions in dedicated schedulers, and resolves conflicts through conflict resolution protocol. It complies with a 3-layer architecture consisting of:

1) Components layer: The bottom layer consists of atomic components. Their interfaces are made of one send-port and one or more receive-ports Components share their lists of enabled ports with the upper layer.

2) Schedulers: The second layer consists of a set of components each hosting a set of interactions. Conflicts between interactions included in the same component are resolved locally. And conflicts between interactions of different scheduler components are resolved using the third layer.

3) Conflict Resolution Protocol (CRP): This layer implements an algorithm based on the idea of message-count technique presented in [13]. It is based on counting the number of times that component participates in an interaction. Conflicts are resolved by ensuring that each participating number is used only once. Different implementations of the reservation protocol are presented in [8].

III. From High-level RT-BIP to TT physical model

In this section, we propose a generic framework for transforming an RT-BIP model into a TT physical model. We first detail subtleties of this transformation with respect to RT-BIP and SR-BIP. Then we present the TT-BIP architecture that addresses these subtleties. Finally we describe how to construct a correct TT-BIP model starting from a high-level RT-BIP model.

A. Subtleties of the transformation

In sequential models, interactions are executed atomically by a single scheduler. To the contrary, introducing TT settings (mainly decomposition into tasks and assuming TT communication mechanism) to this model requires the implementation to deal with more complex issues:

1) Decomposition into Tasks: Tasks (processes, threads ..) are building blocks for TT applications. In Design phase, designers have the choice to model a TT task using one or more BIP components. Thus depending on the user task mapping, tasks should appear in the derived physical model with respect to the initial high-level model.

2) Strong synchronization in BIP interactions Vs. message-passing: In order to respect TT communication setting, the derived physical model should handle intertask communication through dedicated RT-BIP component which stands for the TT communication system. The challenge is to switch from the high-level RT-BIP model, where multiparty synchronized interaction is a primitive, to the TT model, where inter-task communication is performed via a communication medium.

3) Resolving conflicts: In high-level RT-BIP model, conflicts are handled by the single scheduler. TT communication components in the derived model must ensure that execution of conflicting interactions is mutually exclusive.

We address the first issue, by initiating composite task components that encompass atomic components mapped to the same task. The second problem is addressed by breaking the atomicity of execution of interactions, so that a task can execute unobservable actions to notify TT communication component about their states, and then execute the corresponding interaction. Communication between tasks and TT communication component is send/receive interactions. Resolving conflicts leads us to use solution proposed in SR-BIP model, which consists in instantiating a BIP component that implements the algorithm proposed in [13]. The latter uses message counts to ensure synchronization and reduces the conflict resolution problem to dining or drinking philosophers [14].

B. TT-BIP Architecture

In this subsection, we present the TT-BIP architecture that combines elements of RT-BIP, SR-BIP and the TT paradigm to address previously mentioned issues. This architecture is based on *tasks* (processes, threads ..) as one of its building

blocks. A task can be mapped on one or more atomic components. Inter-task communication is handled by dedicated components called Time-Triggered Communication Components (TTCC) standing for schedulers in RT-BIP and SR-BIP models. Conflicts between different TTCC components, are resolved through CRP components of SR-BIP model.

An RT-BIP model B^{TT} complies with the TT-BIP architecture if it consists of three basic entities: Tasks, TTCC and CRP components, organized by the following abstract grammar:

$$
\begin{array}{lll}
TT\text{-}BIP\text{-}Model & ::= & Task^+ \ . \ TTCC^+ \ . \ CRP \ . \ S/R\text{-}connector^+ \\
Task & ::= & atomic\text{-}component^+ \ . \\
 & & atomic\text{-}talking\text{-}component^+ \ . \ connectors^+ \\
TTCC & ::= & TTCC^{NC} \mid TTCC^{C}
\end{array}
$$

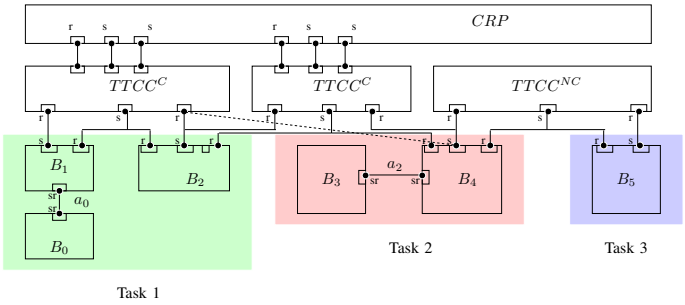


Figure 2. TT-BIP model

Task components (resp. TTCC components) and TTCCs (resp. CRP components) communicate with each other through message-passing, i.e. send/receive interactions. This latter is a set of one send port and one or more receive ports. Communication between components inside a task are classic multi-party RT-BIP interactions (see. Figure 2).

Definition *8:* We say that $B^{TT} = \pi^{TT}\gamma^{TT}(B_1^{TT}, ..., B_n^{TT})$ is a TT-BIP model iff we can partition the set of its interactions in two sets A_I and A_E that are respectively sets of *internal* and *external* interactions, corresponding to intra-task and inter-task interactions, such that:

- Each interaction $a \in A_I$, is a classic multi-party interaction presented by classic RT-BIP connectors relating atomic components inside one task component,
- Ports of Task, TTCC and CRP components can be partitioned into three sets P_u, P_s and P_r that are respectively the set of unary ports, send ports and receive ports. Each interaction $a \in A_E$, is either (1) a send/receive interaction with $a = s, r_1, ..., r_k$, $s \in P_s$, $r_1, ..., r_k \in P_r$ or, (2) a unary interaction $a = p$ with $p \in P_u$,
- The order of execution of transitions labelled by send or receive-ports depends on the nature of component (task, TTCC or CRP component). In task (resp. TTCC and CRP) components, each one or two successive transition(s) labelled by send-port(s) (rep. receive-ports) should be acknowleged by a transition labelled by receive-port (resp. send-port),
- If s is a port in P_s, then there exists one and only one send/receive interaction $a \in \gamma^{TT}$ with $a = (s, r_1, ..., r_k)$ and all ports $r_1, ..., r_k$ are receive-ports. We say that $r_1, ..., r_k$ are receive-ports of s,
- If $a = (s, r_1, ..., r_k)$ is a send/receive interaction in γ^{TT} and s is enabled at some global state of B^{TT}, then all its receive-ports $r_1, ..., r_k$ are also enabled at that state,

The specificity of each constituent element of the TT-BIP architecture is detailed below.

1) Task components: A task component is a composite component consisting of one or more atomic components related to each other using connectors. Atomic components within a task which export their send and receive ports to the task interface are called *atomic-talking-components* (ATC). These latter can only communicate with a TTCC components (through the task component interface). TTCC component interferes only in *external interactions* referring to inter-task interactions A_E. These *external interactions* are presented locally (in the task component) by *artefact interactions*. We denote the set of these artefact interactions by $A_{E_{artefact}}$. The remaining components in a task, do not export their ports. They can only participate in *internal interactions* A_I (see Figure 3).

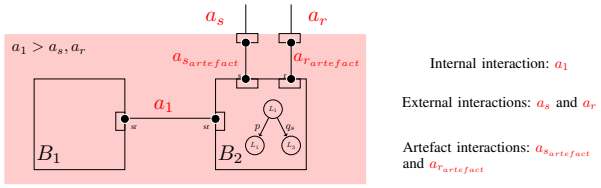


Figure 3. Example of a task component

In task component, conflicts between internal and artefact interactions can be resolved by local priority rules. In such cases, execution of internal interactions are privileged to external ones, that is, internal interactions have more priority than artefact ones (see. Figure 3). Through these priority rules, all possible conflict cases are handled. If the conflict can be resolved locally, the priority rule $a \in A_I > a' \in A_{artefact}$ can be used. Otherwise, the conflict will be handled by TTCC and CRP components.

Each ATC component in the task component has one or more send and receive ports and zero or more standard ports. It contains in addition to observable states and transitions, some partial states and unobservable transitions. An unobservable transition always leads to a partial state. Transitions allowing communication with the TTCC could be seen as a non atomic transition. It starts first by a non visible transition labelled by the send port. It allows to send an offer (i.e. information about active ports) to TTCC component. This transition is followed then by a visible transition labelled by the receive port indicating the completion of the communication. These two transitions are separated by a partial-state location standing for a busy state of the component where it is waiting for a notification from the TTCC via the receive port (cf. Figure 4).

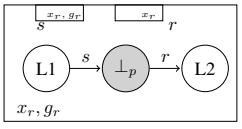
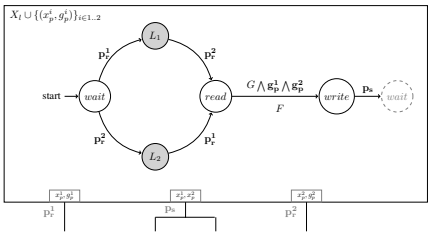


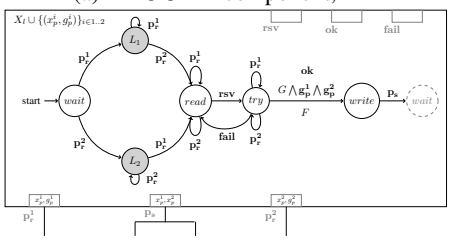
Figure 4. ATC Communication pattern

2) TTCC Component: Each TTCC component represents one interaction $a \in \gamma$ with $a = (a, G_a, F_a)$, we denote

by n the number of components related to TTCC. It is inspired from schedulers of the SR-BIP models implemented for distributed implementations in BIP [9] and RT-BIP [10]. The TTCC component is designed to model the behaviour of the communication system in the TT paradigm in order to execute an interaction a. There are two different types of TTCC components: conflicting and non-conflicting, denoted respectively $TTCC^C$ and $TTCC^{NC}$. TTCC component behaviour is made of three steps; (1) the component reads variables and guards from task components, (2) based on these received guards and the guard of the interaction a, the TTCC component takes a decision by either executing the interaction upon synchronization (i.e. conjunction of read guards evaluates to *True*) if a is a non-conflicting interaction or soliciting the CRP component to find out if the conflicting interaction a can be executed and (3) finally it writes on appropriate task components by sending a notification. Figure 5a presents an example of $TTCC^{NC}$ and Figure 5b presents an example of $TTCC^C$ for $n = 2$.



(a) $TTCC^{NC}$ component, n=2



(b) $TTCC^C$ component, n=2

Figure 5. Conflicting and non conflicting TTCC components

The behaviour of the TTCC components is described as a timed automata. It depends on whether the TTCC handling an interaction a is conflicting ($TTCC^C$) or not ($TTCC^{NC}$).

The set of variables of TTCC can be partitioned into two classes. The first class consists in variables updated whenever an offer from ATC components B_i participating in the interaction a is received. They consist of the following: guard variable g_p^i and a local copy of the variables X_p^i for each port p involved in the interaction a, a time progress condition variable tpc_i and a participation number variable n_i for each ATC component B_i of tasks participating in a. The second class consists in variables updated whenever interaction a is scheduled which are: execution date variable t_{ex_a}, that stores the last execution date of interaction a and, an execution date variable t_{ex_i} for each component B_i participating in a.

In case of a non conflicting TTCC ($TTCC^{NC}$), the set of places contains the followings: For each ATC component B_i within a task involved in the interaction a, we include waiting places: $wait$ and the set $L_\perp$ with $|L_\perp| = \sum_{k=0}^{n-2} \prod_{j=0}^{k} (n-j)$. $L_\perp$ locations present states where component is waiting for reception of other tasks guards and variables. Each place $l_\perp \in L_\perp$ that is reached by a transition receiving offer from B_i, has a time progress condition defined by the variable tpc_i. We include a writing place $write$ that allows notification of tasks (and then ATC components) participating in a. The time progress condition of $write$ is *False*. In case of a conflicting TTCC ($TTCC^C$), the set of states includes the same states of a non conflicting TTCC, with an additional trying state try_a.

The set of ports of a non conflicting TTCC consists of the following: For each ATC component B_i within a task involved in the interaction a,TTCC includes a receive-port p_{r_i}, to receive offers. Each port p_{r_i} is associated with the variables g_p^i and X_p^i for each port p of B_i as well as the variable tpc_i and n_i of B_i. The set of ports of TTCC includes a send-port p_s, which exports the set of variables $\cup_{p \in a} X_p \cup \{t_{ex_i}\}$ where i is the index of ATC component B_i that exports port p. TTCC includes also a unary port a allowing the execution of the interaction a. In case of a conflicting TTCC ($TTCC^C$), the set of ports includes the same ports of a non conflicting TTCC except the unary one, with additional ports; rsv_a associated with the set of variables n_i of components $B_i \in comp(a)$.

Transitions of a non conflicting TTCC are as following: In order to receive data from task components; we include for each $\{l, l'\} \subseteq \{\{wait, read\} \cup L_\perp\}$ and each $i \in [1..n]$ a transition $\tau_{receive_i} = (l, p_r^i, True, Identity, l')$. This transition guard is default to *True*. In order to execute the interaction a we include the Transition leading from state $read$ to state $write$, where $\tau_a = (read, p_u, G_a \bigwedge (\bigwedge_{i=1}^{n} g_p^i), F_a, write)$ where p_u is a unary port, G_a is a guard on variables of the set X_l, and F_a is a function on X_l. To notify task components after executing the interaction a, we include the transition $\tau_{write} = (write, p_s, True, Identity, wait)$.
In case of a conflicting TTCC ($TTCC^C$), the set of transitions includes in addition to transitions $\tau_{receive}$ and τ_{write} of non conflicting TTCC, the following transitions: To each place $l_\perp \in L_\perp$ reached by transition labelled by port p_{r_i} we include a loop transition $\tau_{loop_i} = (l_\perp, p_r^i, True, Identity, l_\perp)$. Before executing interaction a, we include the transition that will solicit the CRP component from the state $read$, where $\tau_{rsv} = (read, rsv, True, Identity, try)$. We also include loop transitions on place $read$, allowing reception from receive ports, such that for all $i \in [1, n]$, $\tau'_{loop_i} = (read, p_r^i, True, identity, read)$. From try location, the first possible transition, depending on the response of the CRP component, $\tau_{ok} = (try, ok, G_a \bigwedge (\bigwedge_{i=1}^{n} g_p^i), F_a, write)$, where G_a is a guard on variables of the set X_l, and F_a is a function on X_l. This transition will execute the interaction a. We also include loop transitions on place try, allowing reception from receive ports, such that for all $i \in [1, n]$, $\tau''_{loop_i} = (try, p_r^i, True, identity, try)$. The second possible transition from state try is $\tau_{fail} = (try, fail, True, Identity, read)$, in that case, a is not allowed to execute.

3) Conflict Resolution Protocol Component: The conflict resolution protocol (CRP) accommodates the algorithm proposed in [13]. It uses message counts to ensure synchronization and reduces the conflict resolution problem to dining or drinking philosophers [14]. Its main role is to check the freshness of requests received for an interaction, that is, to check that no conflicting interactions has been already executed using the same request. In each request, an interaction sends

the participation numbers of its components, i.e., number of interactions each ATC component has participated in. This ensures that two conflicting interactions cannot execute with the same request. Mutual exclusion is ensured using participation numbers. To this end, the conflict resolution protocol keeps the last participation number N_i of each component B_i and compares it with the participation number n_i provided along with the reservation request from TTCC components. If each participation number from the request is greater than the one recorded by the conflict resolution protocol ($n_i > N_i$), the interaction is then granted to execute and N_i is updated to n_i. Otherwise, the interaction execution is disallowed. Figure 6 presents the places, transitions, variables, guards and update functions involved in handling an interaction a with two participating components B_1 and B_2. Whenever a reservation for executing a arrives, this token moves from place $wait_a$ to place $receive_a$. From this place, if the guard of the transition labelled by ok_a is *True*according to the current values of N_i variable and freshly received n_i variables, the transition can take place. The transition labelled by $fail_a$ is always possible.

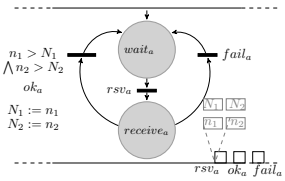


Figure 6. Fragment of the CRP component

C. From High-level RT-BIP model to TT-BIP model

In this section, we present the model transformation from a RT-BIP model $B \stackrel{def}{=} \pi\gamma(B_1, ..., B_n)$ into an equivalent TT-BIP model $B^{TT} = \pi^{TT}\gamma^{TT}(B_1^{TT}, ..., B_n^{TT}, TTCC_1, ..., TTCC_m, CRP)$.

This transformation is parametrized by a user-defined task mapping which consists in associating to each task T_k a group of atomic components of the model B. We denote $\mathcal{B}$ the set of atomic components of model B. We assume, we have $K \leq n$ tasks and we denote by $\mathcal{T} = \{T_k\}_{k \in K}$ the Task set, such that $\mathcal{T}$ is a partition of $\mathcal{B}$: where for all $j, k \in K$ and $j \neq k, T_j \cap T_k = \emptyset$. For all $k \in K$ we have $T_k = \{B_i\}_{i \in I_k}, I_k \subseteq K$ such that $\bigcup_{k \in K} I_k = K$.

The transformation process is performed in two steps as shown in Figure 7. First, an analysis phase allows definition of set of components and connectors to be transformed depending on the task mapping. Then the RT-BIP model is transformed into a TT-BIP model where only inter-task interactions are replaced by TTCC components and intra-task interactions remain intact. Components mapped to the same task are gathered in a composite task component.

We consider that the original RT-BIP model consists only of atomic components and flat connectors. Moreover, each connector defines one and only one interaction. Indeed, these assumptions do not impose any restrictions on the original model, since we can apply the transformations "Component flattening" and "Connector flattening" of previous research work in [15].

D. Analysis Phase

Starting from a high-level RT-BIP model and a user task mapping, a TT-BIP model can be derived. Thus, we have first

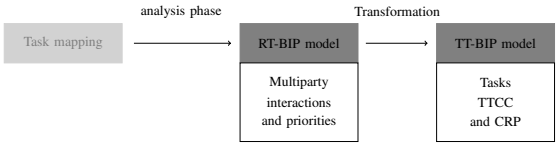


Figure 7. Transformation from RT-BIP to TT-BIP model

to identify internal and external interactions as well as ATC components. Conflicts between external interactions are held by CRP components. But conflicts between an internal and external interactions should be also resolved. We distinguish between three cases of a conflict between an internal and external interaction; (1) conflict not resolved in the initial model by a priority rule, (2) conflict initially resolved by a static priority rule and (3) a conflict initially resolved by a dynamic priority rule. These three cases should be taken into account. In the first case, the conflict can be resolved locally in the task by imposing a priority rule that privileges the internal interaction (see Figure 3). In the second case, the conflict is already resolved and the initial priority rule is kept in the derived model. In the third case, task component can't resolve the conflict locally, since it doesn't have access to the global state of the model and thus can't evaluate the dynamic priority rule. In that case, the internal conflicting interaction is exported to be held by a dedicated TTCC. This interaction is thus being an external interaction. And the conflict is being a conflict between two external interactions. It can now be resolved by the CRP component initially intended to handle such a conflict.

The algorithm of the analysis phase initialises sets of internal, external and ATC components (A'_I, A'_E and B'^{ATC}) based on the task mapping. And then, it defines new sets (A_I, A_E and B^{ATC}) depending on conflicts between internal and external interactions and existing priority rules related to this conflict. These obtained sets are inputs for the transformation process.

A'_I, A'_E and B'^{ATC} sets are defined depending on the user task mapping as follows:

- External interactions: A'_E set is defined as the set of interactions in the participant components of which we can find at least two different atomic components belonging to two different tasks.
  Formally, $A'_E = \{a \in \gamma \mid \exists B_1, B_2 \in comp(a), T_1, T_2 \in \mathcal{T} : B_1 \in T_1, B_2 \in T_2, T_1 \neq T_2\}$.
- Internal interactions: A'_I set in defined as the set of interactions in the participant components of which we can find only atomic components belonging to the same task. $A'_I = \gamma \setminus A'_E$.
- Atomic talking components: $\mathcal{B}'^{ATC}$ set is initialised to the set of atomic components in $\mathcal{B}$ that are concerned by external interactions (i.e. are related to other atomic components belonging to a different task). We define $\mathcal{B}'^{ATC} = \{B \in \mathcal{B} | A'_E \cap P_B \neq \emptyset\}$, where P_B is the port set of the component B.

After defining initial sets of internal interactions, external interactions and ATC components, the final sets A_E, A_I and $\mathcal{B}^{ATC}$ are defined depending on existing priority rules following Rule 1.

***Rule** 1:* For each $a \in A'_I$, and, $a' \in A'_E$ such that $\exists q \in Q$, $(C(Q), a, a') \in \pi_Q$, $A_E = A'_E \cup \{a\}$, $A_I = \gamma \setminus A_E$, $\mathcal{B}^{ATC} = \{B \in \mathcal{B} | A_E \cap P_B \neq \emptyset\}$.

E. Transformation rules

Starting from an RT-BIP model $B \stackrel{def}{=} \pi\gamma(B_1, ..., B_n)$, we first apply a variation of the transformation from RT-BIP to SR-BIP [8] which transforms original atomic components defined in the RT-BIP model by send/receive components, handles initial interactions using schedulers and adds a conflict resolution protocol component for conflict resolution. Then we instantiate components associated to the same task in one composite task component following Rule 4.

In our case, we need to implement intertask communication using dedicating components instead of BIP connectors. Thus we propose here to apply the send/receive transformation only to the subset of components and connectors dedicated to inter-task communication in the initial model.Thus, only ATC components are transformed in send/receive atomic components following Rule 2, and only inter-task connectors referring to external interactions A_E are replaced by TTCC components and the CRP component following Rule 3.

1) ATC component transformation: Every atomic component can define a set of local clocks. They can be reset at any time and are involved in timing constraints and time progress conditions. TTCC component makes use only of the common time scale presented by a global clock C^gwhich is initialized to 0 and is never reset, and measures the absolute time elapsed since the system started executing.

Since TTCC receives timing constraints and time progress conditions from different ATC components, it would be better if these latter transforms their sent data using the global clock. Therefore, we follow the approach of [6]: for each clock c of an atomic component B, we introduce a variable ρ_c that stores the absolute time of the last reset of c. This variable is updated whenever the component executes a transition resetting clock c. In fact, when the TTCC executes an interaction a, it stores its execution date in a variable t_{ex_a}. The value of this variable is then sent to participating components (during the notification). Each participating component executes then the corresponding transition according to the received notification, and updates each variable ρ_c to t_{ex_a} if the transition resets clock c in the original model. Notice that the value of c can be computed from the current value of C^g and ρ_c by using the equality $c = C^g - \rho_c$. Using ρ_c, any timing constraint tc involved in a component B can be expressed using the clock C^g instead of clocks C. We can transform tc as follows:

$$tc = \bigvee_{k=1}^{m} \bigwedge_{c \in C} l_c^k \leqslant c \leqslant u_c^k = \bigvee_{k=1}^{m} \bigwedge_{c \in C} l_c^k + \rho_c \leqslant C^g \leqslant u_c^k + \rho_c \tag{5}$$

Similarly, any time progress condition tpc involved in B is transformed using the clock C^g as follows:

$$tpc = \bigvee_{c \in C} c \leqslant u_c = \bigvee_{c \in C} C^g \leqslant u_c + \rho_c \tag{6}$$

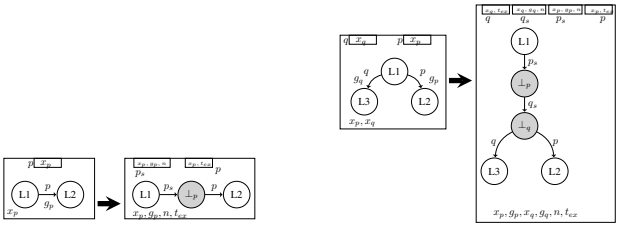
We transform then an ATC component $B^{ATC} = (L, P, X, C, T, \{tpc_l\}_{l \in L})$ of a RT-BIP model into a TT ATC component $B^{TT} = (L^{TT}, P^{TT}, X^{TT}, C^{TT}, T^{TT}, \{tpc_l^{TT}\}_{l \in L})$ that is capable of communicating with $TTCC$ component(s). We denote by a the initial interaction between two or more ATC components and by p the port of B^{ATC} such that $p \in a$. Instead of the initial connector linking these atomic components, communication is performed through added $TTCC$ component. Thus each atomic component will be connected to $TTCC$ component using send/receive connectors.

***Rule** 2:* In each ATC component $B^{ATC} \in \mathcal{B}^{ATC}$, for each port $p \in P \cap A_E$, each transition labelled by the port p will be duplicated in two transitions with an intermediate partial state $\perp_p$. A new added send-port p_s and the port p (Receive port) are the successive labels of the two new transitions. The transition labelled by p_s is an unobservable transition. Formally, B^{TT} is obtained from B as follows:

- $L^{TT} = L \cup L_\perp$, where $L_\perp = \{\perp_{l,p} \ | \exists \tau = (l, p, g_\tau, f_\tau, r, l') \in T, a \in A_E, p \in P \cap a\}$,
- $P^{TT} = P \cup P_s$, where $P_s = \{p_s | p \in P \cap A_E\}$,
- $X^{TT} = X \cup \{g_p\}_{p \in P \cap A_E} \cup \{nb, \rho_c, t_{ex_a}\}$, where nb is the number of participation of B^{TT},
- $C^{TT} = C, C^g$,
- $T^{TT} = T \backslash \{\tau_p\}_{p \in P \cap A_E} \cup \{\tau_{p_s}, \tau'_p\}_{p \in P \cap A_E}$ where $p \in P \cap A_E$, $\tau_p \in T$ and $\tau_{p_s}, \tau'_p \in T^{TT}$ such that $\tau_p = (l, p, g_{\tau_p}, f_{\tau_p}, l')$, $\tau_{p_s} = (l, p_s, True, g_p = g_{\tau_p}, L_\perp)$ and $\tau'_p = (L_\perp, p, True, f_{\tau'_p}, l')$. $f_{\tau'_p}$ executes f_{τ_p} and increments nb.
- For places $L_\perp$, the time progress condition is $tpc_{L_\perp}^{TT} = True$.

We denote the set of transformed ATC components by ${\mathcal{B}_{\mathcal{ATC}}}^{TT}$.

Figure 8 illustrates this transformation for the cases of conflicting and non conflicting interactions.



(a) Non conflicting interaction (b) Conflicting interaction

Figure 8. Atomic Component transformation

2) Connector transformation: Each connector presenting an inter-task communication will be replaced by a TTCC component as described by the following rule:

***Rule** 3:* Let C_k be a connector defining an external communication between k components of different tasks which handles an external interaction $(a, X_a, G_a, F_a) \in A_E$. We replace C_k by a $TTCC_k$ component and a set of corresponding connectors, as follows::

- if a is a conflicting interaction (i.e. $\exists a' \in A_E$ such that $a \# a'$), $TTCC_k$ is an instance of $TTCC^C$ components, else $TTCC_k$ is an instance of $TTCC^{NC}$ component.
- In both cases, the set of local variable of $TTCC_k$ is $X_l = X_a \cup \{nb, t_{ex}\}$, and in the transition executing the interaction a, we have the guard $G = G_a \bigwedge (\bigwedge_{i=1}^{k}) g_p^i$ and the function $F = F_a$.
- k binary and one k+1$-ary$ send/receive connectors are instantiated to connect $TTCC$ to ATC components,
- A CRP component is then instantiated and connected to $TTCC^C$ components via three binary send/receive connectors each.

3) Task component instantiation: Task components are created according to the task mapping. Each transformed ATC component exports its send and receive ports to the task component interface. Each exported port will be referenced by an artefact external interaction $a_{artefact}$ that presents locally the external interaction involving the task. Conflicts between internal and artefact interactions that are not initially resolved by priority rules, are resolved by a static added rule that favours the internal interaction.

***Rule** 4:* For each task $T_k = \{B_i\}_{i \in I_k} \in \mathcal{T}$, we instantiate a composite component T_k^{TT} where:

- The set of exported ports of T_k^{TT} is $P_T^{TT} = (P_s \cap \bigcup_{B_i^{TT} \in T_k^{TT} \cap \mathcal{B}_{ATC}{}^{TT}} P_{B_i^{TT}}) \cup (P_r \cap \bigcup_{B_i^{TT} \in T_k^{TT} \cap \mathcal{B}_{ATC}{}^{TT}} P_{B_i^{TT}})$,
- for each interaction $a \in A_I$ in conflict with an interaction $b \in A_E$, we add the priority rule $\pi = (\mathit{True}, b, a)$.

In this section we proposed TT-BIP architecture as a TT physical model that allows inter-task interactions only through a communication media (TTCC components) in order to comply with the TT paradigm. We also defined a transformation process that derives correctly a TT-BIP model from a high-level RT-BIP model. The obtained TT-BIP model is intended to be translated into the language specific to target TT platform. Since each operating system (OS) offers specific services (communication, scheduling policy...), it is advantageous to define a mapping between these offered services and their corresponding elements in the TT-BIP model. To be able to do such a mapping, we consider the decomposition of the TTCC component into an equivalent set of atomic components each describing a step of its behaviour: components allowing individual acquisition of offers, components to aggregate them, and component executing the interaction. We believe that some parts of TTCC subcomponents can be mapped into OS communication services. Priorities in the TT-BIP model (existing and added one in task component) are priorities on interactions. In simulation level they can be resolved by RT-BIP sequential scheduler. In order to be handled in implementation, these priority rules should be translated either into predicates on transitions following [9], or, into priorities on tasks, which could parametrize the scheduler of the target platform. This adequation between subset of TTCC (resp. priority rules) and OS communication services (resp. schedulers) will be the object of a future publication.

IV. Case Study

In this section, we present our case study which consists in a Flight Simulator (FS) application. We have first modelled the FS application in RT-BIP starting from FS Modelica model. And then we applied manually the transformation to derive the equivalent TT-BIP model. The simulation of three models reveals that their outputs are strictly similar.

The initial RT-BIP model, consists of a set of six communicating components (see. Figure 9); autopilot, fly-by-wire, route, servo, simulator and sensor. The autopilot models the pilot commands in function of the flight state; it has four main functionalities; flight state reception from sensor component, execution of the route planner, execution of fly-by-wire and sending command to servo component. The route component computes distance to current waypoint and change route towards next waypoint if necessary. It operates in low frequency: every 15 seconds. The fly-by-wire component allows course correction by setting roll attitude and ailerons and elevator. It operates in high frequency: every 5 seconds. The servo refers to the autopilot's actuation on plane's flight control surfaces. Servo component receives command from autopilot component and transfers it to simulator component. The Flight simulator simulates flight dynamics computation of plane and wing tips position based on received commands (i.e. new values of roll, pitch and throttle). The sensor refers to the autopilot's perception of real world data. Sensor component receive data about flight state from simulator component and resend them to the autopilot component. This component can add some noise, delay, etc. to mimic realistic data acquisitions. But in our example, we stand for copying the state computed by simulator component.

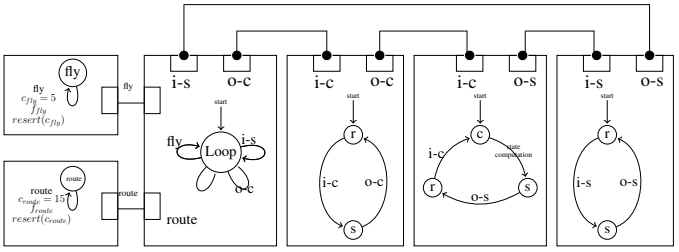


Figure 9. Initial Flightsim model

We apply the transformation and derive the TT-BIP model. The chosen task mapping is as follows; the first task compromises autopilot, route and fly-by-wire components. Servo, sim and sensor are each mapped on a different task. Thus only connectors route and fly remain intact. Other connectors are inter-task connectors. They are transformed in TTCC components. The interactions connecting the autopilot component to the sensor and servo components are conflicting in the state "loop" of the autopilot. Thus instantiated TTCC for these interactions are related to CRP component. Figure 10) shows the obtained model. Behaviours of TTCC and CRP components is not displayed for reason of space. Nonetheless, since all TTCC component are connecting exactly two tasks, their automata are similar to those in 5.

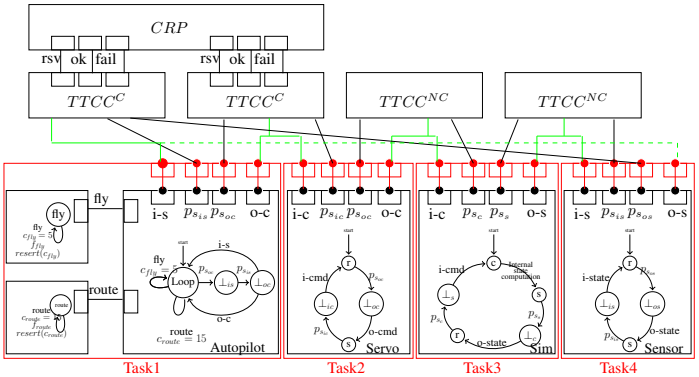


Figure 10. Final Flightsim model

In order to be able to compare functionality of two models, we generate C++ code from the initial and the obtained model. Simulation of two generated codes, allowed us to visualize and compare the output signals. A band shows the trajectories of left and right wingtips and illustrates the roll movement that precedes the change in course at each waypoint, while

the plane progressively reaches its desired altitude. Figure 11 presents the simulation results of the initial and the derived models, for the waypoints (300,0,300), (300,300,300), (0,300,300) and (0,0,300). Visual inspection reveals that the output of the transformed model is strictly similar to that of the original model (see Figure 11).

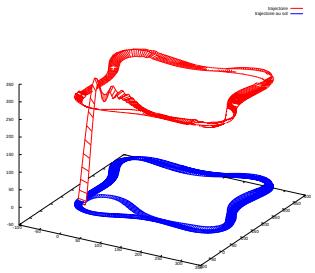


Figure 11. Trajectories of left and right wingtips

V. Related work

There have been a number of approaches exposing the relevant features of the underlying architectures at high level design tool.

Authors in [16] present a design framework based on UML diagrams for applications running on Time Triggered Architecture(TTA). This approach doesn't support earlier architectural design phase and needs a backward mechanisms for the generated code verification. It also doesn't target generic TT implementations since it assumes the underlying TT protocol to be The Flexray standard.

Authors of [17] and [18] present a method to reduce the gap between models used for timing analysis and for code generation. This method relies on AADL model transformations in order to lower automatically the abstraction level of models. However, this work did not rely on formal semantics.

Since BIP design flow is unique due to its single semantic framework used to support application modelling and to generate correct-by-construction code, many approaches tend to use it to translate high level models into physical models including architectural features. For instance, in [15], a distributed BIP model is generated from a high level one. In [19], a method is presented for generating a mixed hardware/software system model for many-core platforms from an application software and a mapping. These two approaches take advantages from BIP framework but they do not address the TT paradigm.

To the best of our knowledge, our approach is the first to address the problem of deriving progressively a TT physical model in a single host component-based language rooted in well defined semantics.

VI. Conclusion

In this paper, we proposed a chain of transformations that starts from an arbitrary abstract RT-BIP model, and a user-defined task mapping. The transformation process obtains a model that suits the TT-BIP architecture and consists of: (1) breaking atomicity of actions in ATC components by replacing strong synchronizations with asynchronous send/receive interactions, (2) inserting TTCC components that coordinate execution of inter-task interactions according to a user-defined task mapping, (3) extending the model with an algorithm for handling conflicts between TTCC and (4) adding local priority rules in task components for handling conflicts between inter-task and intra-task interactions. We have shown correctness of the final model by trace inclusion.

For future work, we plan to automate the transformation process in RT-BIP modelling environment. Furthermore, we are working on the tool allowing code generation for the target TT platform, that depends on the offered communication services of target operating system. For each specific target operating system, it translates an adapted version of the model.

References

[1] H. Kopetz, "The time-triggered approach to real-time system design," Predictably Dependable Computing Systems. Springer, 1995.

[2] ——, "Time-triggered real-time computing," Annual Reviews in Control, vol. 27, no. 1, 2003, pp. 3–13.

[3] W. Elmenreich, G. Bauer, and H. Kopetz, "The time-triggered paradigm," in Proceedings of the Workshop on Time-Triggered and Real-Time Communication, Manno, Switzerland, 2003.

[4] BIP2 Documentation, July 2012.

[5] T. Abdellatif, "Rigourous implementation of real-time systems," Ph.D. dissertation, UJF, 2012.

[6] T. Abdellatif, J. Combaz, and J. Sifakis, "Model-based implementation of real-time applications," May 2010, pp. 229–238.

[7] R. Alur and D. L. Dill, "A theory of timed automata," Theoretical computer science, vol. 126, no. 2, 1994, pp. 183–235.

[8] B. Bonakdarpour, M. Bozga, M. Jaber, J. Quilbeuf, and J. Sifakis, "From high-level component-based models to distributed implementations," in Proceedings of the tenth ACM international conference on Embedded software. ACM, 2010, pp. 209–218.

[9] J. Quilbeuf, "Distributed implementations of component-based systems with prioritized multiparty interactions. application to the bip framework." Ph.D. dissertation, Université de Grenoble, 2013.

[10] A. Triki, B. Bonakdarpour, J. Combaz, and S. Bensalem, "Automated conflict-free concurrent implementation of timed component-based models," in NASA Formal Methods. Springer, 2015, pp. 359–374.

[11] R. Alur and D. L. Dill, "A theory of timed automata," Theoretical computer science, vol. 126, no. 2, 1994, pp. 183–235.

[12] A. Basu, P. Bidinger, M. Bozga, and J. Sifakis, "Distributed semantics and implementation for systems with interaction and priority," in Formal Techniques for Networked and Distributed Systems–FORTE 2008. Springer, 2008, pp. 116–133.

[13] R. Bagrodia, "Process synchronization: Design and performance evaluation of distributed algorithms," Software Engineering, IEEE Transactions on, vol. 15, no. 9, 1989, pp. 1053–1065.

[14] K. M. Chandy and J. Misra, "The drinking philosophers problem," ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 6, no. 4, 1984, pp. 632–646.

[15] M. Bozga, M. Jaber, and J. Sifakis, "Source-to-source architecture transformation for performance optimization in bip," Industrial Informatics, IEEE Transactions on, vol. 6, no. 4, 2010, pp. 708–718.

[16] K. D. Nguyen, P. Thiagarajan, and W.-F. Wong, "A uml-based design framework for time-triggered applications," in Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International. IEEE, 2007, pp. 39–48.

[17] E. Borde, S. Rahmoun, F. Cadoret, L. Pautet, F. Singhoff, and P. Dissaux, "Architecture models refinement for fine grain timing analysis of embedded systems," in Rapid System Prototyping (RSP), 2014 25th IEEE International Symposium on. IEEE, 2014, pp. 44–50.

[18] F. Cadoret, E. Borde, S. Gardoll, and L. Pautet, "Design patterns for rule-based refinement of safety critical embedded systems models," in Engineering of Complex Computer Systems (ICECCS), 2012 17th International Conference on. IEEE, 2012, pp. 67–76.

[19] P. Bourgos, "Rigorous design flow for program-ming manycore platforms."

[20] R. Milner, Communication and Concurrency. Hertfordshire, UK, UK: Prentice Hall International (UK) Ltd., 1995.

APPENDIX
CORRECTNESS PROOFS

In this section we prove the correctness of our TT-BIP model. First, we show that the obtained model is indeed a TT-BIP model. Second, in order to prove the correctness of our transformation, we consider these two steps in transforming a model B; the first step that transforms ATC components and inter-task connectors, and the second step that instantiates composite task components with priorities. We denote by B_{SR}^{TT} the output model of the first step and by B^{TT} the final model after applying the second step. We show that B weakly simulates B_{SR}^{TT}. And finally we prove trace inclusion between B_{SR}^{TT} and B^{TT}, i.e., the second step of transformation preserves safety property.

A. Compliance with TT-BIP model

We need to show that receive ports of B^{TT} model will unconditionally become enabled whenever one of the corresponding send ports is enabled. Intuitively, this holds since communications between tasks and TTCC components, and between TTCC components and CRP component follow a request/acknowledgement pattern. Whenever a component sends a request (via a send port) it enables the receive port to receive acknowledgement.

In ATC components, we denote by $l_\perp$ each place in L^{TT} that precedes a $\perp_p$ place. And by l other places. In AC components, all places are denoted by l.

***Lemma** 1:* Given an RT-BIP model $B = \pi\gamma(B_1, ..., B_n)$ and a task mapping $T = \{T_1, ..., T_k\}$, the model $B^{TT} = \pi^{TT}\gamma^{TT}(B_1^{TT}, ..., B_n^{TT}, TTCC_1, ..., TTCC_m, CRP)$ obtained by transformation of section III-C meets the properties of definition 8.

Proof:

The first four constraints of Definition 8 are trivially met by construction. We now prove that the fifth constraint also holds, i.e., whenever a send port is enabled, all its associated receive ports are enabled as well.

- Between a task component B_i^{TT} and a $TTCC_j^{NC}$ component, for all interactions involving a component B_i, we distinguish between 3 classes of states:
  - The first class contains all states between wait and Read states of $TTCC_j$ where this latter is enabling all its receive ports in all possible orders and B_i^{TT} is in a busy location $l_\perp$. This class presents waiting states. From that class, the only enabled send-port involved in an interaction with B_i^{TT} is the port p_{si} of B_i. By definition of the class, all associated receive ports are also enabled, and the send/receive interaction can take place to reach a state of the second class.
  - In the second class, the component B_i^{TT} is either in a place $\perp_p$ that is not a busy location, and the TTCC component is in the Read place. From that configuration, there is no enabled send-port involved in an interaction with B_i^{TT}. The next class of states is reached when the TTCC executes a transition corresponding to an interaction involving B_i.
  - In the third class, the component B_i^{TT} is in place l. From this state only internal interactions could be enabled and no communication with TTCC component can be planned (no send-port is active).
  - In the remaining class, the TTCC component is in the place write. The port p_s of TTCC is enabled. By construction, the component B_i^{TT} sends an offer from l for port p only if the receive port p_{ri} is enabled from place $\perp_p$ in B_i^{TT}. Thus the send/receive interaction can take place to reach back the first class of states.
- Between a task component B_i^{TT} and a $TTCC_j^C$ component, for all interactions involving the interaction a which is externally conflicting, we have almost the same four classes as non conflicting TTCC. $TTCC_j^C$ refers to CRP before executing a. It uses the current participation number of B_i for the execution of a and no other interaction is granted using the same participation number. Thus, write is the only active place, from which a notification could be sent to a component B_i.
- Between the TTCC component $TTCC_j^C$ and the conflict resolution protocol CRP, we consider the places try_a in the component $TTCC_j^C$, $wait_a$ and $treat_a$ in CRP component. If $TTCC_j^C$ in not in try_a place, and CRP in in the $wait_a$ place, only reservation request through port rsv_a is enabled. When the rsv_a request is sent, the place try_a and $treat_a$ become active. From this configuration, only send ports ok_a and $fail_a$ are enabled in the CRP, and the associated ports are also enabled in the TTCC.

■

This proof ensures that any component ready to perform a transition labelled by a send-port will not be blocked by waiting for the corresponding receive-ports.

B. Observational Equivalence between B and B_{SR}^{TT}

We denote by B the initial model and by B_{SR}^{TT} the resulting model of the step 1 of the transformation.

We show that B and B_{SR}^{TT} are observationally equivalent. The definition of observational equivalence between two transition systems $A = (Q_A, P_A \cup \{\beta\}, \underset{A}{\longrightarrow})$ and $B = (Q_B, P_B \cup \{\beta\}, \underset{B}{\longrightarrow})$ is based on the usual definition of weak bisimilarity [20], where β-transitions are considered unobservable.

***Definition** 9:* (Weak Simulation A weak simulation over A and B, denoted $A \subset B$, is a relation $R \subset Q_A \times Q_B$, such that: $\forall(q, r) \in R$, $a \in P$: $q_A \overset{a}{\rightarrow}_A q' \implies \exists r'$: $(q', r') \in R \wedge r \overset{\beta^* a \beta^*}{\rightarrow}_B r'$ and $\forall(q, r) \in R$: $q \overset{\beta}{\rightarrow}_A q' \implies \exists r'$: $(q', r') \in R \wedge r \overset{\beta^*}{\rightarrow}_B r'$.

A weak bisimulation over A and B is a relation R such that R and R^{-1} are both weak simulations. we say that A and B are *observationally equivalent* and we write $A \sim B$ if for each state of A there is a weakly bisimilar state of B and conversely.

We consider the correspondence between actions of B and B_{SR}^{TT} as follows. To each interaction $a \in \gamma$ of B, we associate either the multi-party interaction a_{MP}, the binary interaction ok_a or the unary interaction a of B_{SR}^{TT}, depending on whether a is internal, external conflicting or external and not conflicting interaction. Other interactions of B_{SR}^{TT} (send/receive interactions) are unobservable and denoted by β.

We proceed as follow to complete the proof of observational equivalence. Among unobservable actions β, we distinguish between β_1 actions, that are interactions between ATC components and TTCC components, and β_2 actions that are interactions between TTCC components and CRP component

(namely the reserve and fail). We denote by q_{SR}^{TT} a state of B_{SR}^{TT} and q a state of B. A state of B_{SR}^{TT} from where no β_1 action is possible is called a stable state, in the sense that any β action from this state does not change the state of atomic components.

In ATC components, we denote by $l_\perp$ in L^{TT} each place allowing sending offer to TTCC component. Note that $l_\perp$ precedes one $\perp_p$ place when only one intertask interaction is possible (e.g. the place L_1 in the right part of Figure 8a). It consists in all states allowing to send successive offers to TTCC components in case of conflicting intertask interactions (e.g. the places L_1 and $\perp_p$ in the right part of Figure 8b). We denote by $\perp_p^*$ each $\perp_p$ place from which no offer could be sent (e.g. the place L_p in the right part of Figure 8a and the place L_q in the right part of Figure 8b). We denote by l places that are neither $l_\perp$ nor $\perp_p^*$ places. In AC components, all places are denoted by l.

***Lemma** 2:* From any state q_{SR}^{TT}, there exists a unique stable state $[q_{SR}^{TT}]$ such that $q_{SR}^{TT} \rightarrow_{\gamma^{TT}}^{\beta_1^*} [q_{TT}^{SR}]$

Proof: The state $[q_{SR}^{TT}]$ exists since in AC component no β transitions are possible and since each ATC component $B_{SR\,i}^{TT}$ can either send one or successive offers, or receive a notification. Since two β_1 transitions involving two different components are independent (i.e. modify distinct variables and places), the same final state is reached independently of the order of execution of β_1 actions. Thus $[q_{SR}^{TT}]$ is unique. ■

The above lemma proves the existence of a well-defined stable state for any of the transient states reachable by the B_{SR}^{TT} model. The state $[q_{SR}^{TT}]$ verifies the property $q_{SR}^{TT} \rightarrow_{\gamma^{TT}}^{\beta_1^*} [q_{TT}^{SR}]$ and $[q_{SR}^{TT}] \not\rightarrow_{\gamma^{TT}}^{\beta_1^*}$.

***Lemma** 3:* At a stable state $[q_{SR}^{TT}]$, the B_{SR}^{TT} model verifies the following properties:

- All ATC components are in non busy places $\perp_p^*$ or l.
- All other atomic components are in a non busy place l.
- All TTCC components are in receive places $read$,
- The clock C^g and all variables in ATC components have the same value than their copies in the TTCC component.

Proof: The three first points come from Lemma 1 that guarantees possible execution of a send/receive interaction if its send-port is enabled. Therefore no place $write$ in the TTCC components (respectively $l_\perp$ in atomic components) can be active at $[q_{SR}^{TT}]$, otherwise the answer p_s of TTCC (respectively the offer from $l_\perp$) could occur. Furthermore, since all offers have been sent, none of the TTCC components can be in $wait$ place. In each $TTCC_j$, when $read$ place is active, the last executed transitions are either offers or a fail message reception. The latter does not modify variables in the $TTCC_j$. For each variable x in the $TTCC_j$, the last modifying transitions are offers from the corresponding atomic components B_i, which ensures that each variable in the TTCC component has the same value as the corresponding ATC component. The clock C^g has the same value in the atomic components and the TTCC as it is never reset. ■

***Lemma** 4:* When ATC component $B_{SR\,i}^{TT}$ is in a stable state, we have $n_i > N_i$

Proof: Where in a stable state, the ATC component is either in place $\perp_p^*$ or l following Lemma 3. When ATC component is in place $\perp_p^*$ all offers have been sent, thus the participation numbers in TTCC corresponds to those in components. Initially, for each ATC component B_i, $N_i = 0$ and $n_i = 1$. By letting all components sending offers to all TTCC components, we reach the first stable state where the property holds, since β_1 actions do not modify the N_i variables. The variables N_i in the CRP component are updated upon execution of an ok_a transition, using values provided by the TTCC, that are values from components according to Lemma 3. Thus, in the unstable state reached immediately after an ok_a transition, we have $n_i = N_i$ for each component $B_{SR\,i}^{TT}$ participant in a. Then, the notification transition increments participation numbers in components so that in the next stable state $n_i > N_i$. For components B_i' not participating in a, by induction on the number of $okinteractions$, we have $n_i' > N_i'$. Now when ATC component is in place l, only multiparty interaction is possible with AC components. This interaction involves only AC components, and thus has no effect on N_i, it only increments n_i. If a previous inter-task interaction has been performed so at state l, we have $n_i > N_i$. If no intertask interaction has been performed before reaching place l, the property holds since initially $n_i = 1$ and $N_i = 0$, and since each action from a state l increments n_i. ■

Lemma 4 shows that the participation numbers propagate in a correct manner. In particular, at any stable state the conflict resolution protocol has only previously used values and TTCC components have the freshest values, that is the same as in ATC components. To prove the correctness of the step 1 of our transformation, we exhibit a relation between the states Q of the original model $B = \{B_1, .., B_n\}$ and the states Q_{SR}^{TT} of final model $B_{SR}^{TT} = \{B_{SR\,1}^{TT}, .., B_{SR\,n}^{TT}\}$ and prove that it is an observational equivalence.

We define the relation by assigning to each state$q_{SR}^{TT} \in Q_{SR}^{TT}$ an equivalent state $equ(q_{SR}^{TT})$ Q by:

1) considering the unique stable state $[q_{SR}^{TT}]$ reachable by doing β transitions.
2) considering the control location $l_\perp$ and $\perp_p^*$ in $B_{SR\,i}^{TT}$ as the control location l for B_i, in $equ(q_{SR}^{TT})$. The control location l in $B_{SR\,i}^{TT}$ are considered as the control location l for B_i, in $equ(q_{SR}^{TT})$. Lemma 3 ensures that it is a valid control state for B_i.
3) taking the valuation of variables of $B_{SR\,i}^{TT}$ to a valuation of variables in B_i, and
4) taking the valuation of original clock c_i in B_i as the valuation of $g_{\rho_{c_i}}$.

***Theorem** 1:* $B_{SR}^{TT} \sim B^{TT}$

Proof: We then define the equivalence R by taking:

$$R = \{(q_{SR}^{TT}, q) \in Q_{SR}^{TT} \times Q | q = equ(q_{SR}^{TT})\} \tag{7}$$

The three next assertions prove that R is a weak bisimulation:

1) If $(q_{SR}^{TT}, q) \in R$ and $q_{SR}^{TT} \xrightarrow{\beta} r_{SR}^{TT}$ then $(r_{SR}^{TT}, q) \in R$.
2) If $(q_{SR}^{TT}, q) \in R$ and $q_{SR}^{TT} \xrightarrow{\sigma} r_{SR}^{TT}$ then $\exists r \in Q : q_{SR}^{TT} \xrightarrow{\sigma} r$ and $(r_{SR}^{TT}, r) \in R$.
3) If $(q_{SR}^{TT}, q) \in R$ and $q \xrightarrow{\sigma}_\gamma r$, then $\exists r_{SR}^{TT} \in Q_{SR}^{TT}$: $q_{SR}^{TT} \xrightarrow{\beta * \sigma}_{\gamma_{SR}^{TT}} r_{SR}^{TT}$ and $(r^{TT}, r) \in R$.

The property 1) is a direct consequence of Lemma 2. If $q_{SR}^{TT} \rightarrow_{\gamma_{SR}^{TT}}^{\beta} r_{SR}^{TT}$, then β is either β_1 action, thus by definition

$[q_{SR}^{TT}] = [r_{SR}^{TT}]$, or β is β_2 action which does not change the state of the atomic components, thus $[q_{SR}^{TT}] = [r_{SR}^{TT}]$. Thus, we have $equ(q_{SR}^{TT}) = equ(r_{SR}^{TT})$.

To prove property 2), we assume the action σ in B^{TT} is either an intra-task interaction (multi-party interaction) a, or an inter-task interaction (corresponding to a unary oy binary interaction) or a delay step δ.
If a is an intra-task interaction, it is not concerned by the transformation. That is $q = equ(q_{SR}^{TT}$ and $r = equ(r_{SR}^{TT}$.
If a is an inter-task interaction, it corresponds to executing a transition labelled by a unary port a in TTCC component handling a or a transition labelled by ok_a. These transitions are enabled according to valuations of variables and clock C^g in the TTCC handling a. If a is not externally conflicting, by construction of the TTCC, the transition labelled by a has the conjunction of guards g_p sent by the ATCs of different tasks for each $p \in a$. Thus the guard of this transition is $G = G_a \bigwedge_{p \in a}(\bigwedge) g_p$, where $g_p = g_p^X \bigwedge tc_p$ being the conjunction of the boolean guard over variables, and timing constraints. By lemma 3 these values are the same in atomic talking components (ATC), and by extension in $q = equ(q_{SR}^{TT})$. Thus the guard of a evaluates to true at $q = equ(q_{SR}^{TT})$. By construction of ATC components, the guard $g_p = g_p^X \bigwedge tc_p$ sent from $l_\perp$ in $B_{SR\,i}^{TT}$ are boolean guards and timing constraints at state l in B_i.
These timing constraints are expressed on clock C^g which could be equivalently expressed on original clocks c involved in the original timing constraints of B_i. The valuation of each clock c involved in the timing constraint of a is computed from the valuation of $g - \rho_c$ where ρ_c is the last clock reset date of c. Therefore, at state q the timing constraint of a expressed on its original clocks are also met.
if a is externally conflicting, the transition labelled ok_a in CRP is possible only if the transition rsv_a executes in the TTCC component. This transition has the same guard and timing constraint of transition labelled by a. Thus, if this transition is possible in the TTCC component, then the guard of a is met at q. Moreover, if transition labelled ok_a is enabled, this means that for each component B_i involved in a, $n_i > N_i$. In particular, for each involved component B_i, the offer corresponding to the number n_i has not been consumed yet. Thus, we conclude that in both cases, we have $q \xrightarrow{a} r$. Finally, executing a in B_{SR}^{TT} triggers the execution of the data transfer function F_a, followed by the computation in ATC upon reception of the response. Thus at $[r_{SR}^{TT}]$, the values in ATC components are the same as in r, which yields $(r_{SR}^{TT}, r) \in R$.

if a is a delay step, it corresponds by letting time progress by δ in either busy locations $l_\perp$ or in places between $wait$ and $read$ of the TTCC components, corresponding to ATC components B_i. Location $l_{\perp i}$ has the time progress condition tpc_{l_i} and states between $wait$ and $Read$ in TTCC has each the time progress condition tpc_i sent from the ATC components and corresponding to time progress condition of location l_i. Thus, all these time progress conditions are not false, otherwise the δ delay step would not be allowed.

By Lemma 3, the values involved in time progress condition and sent from l_perp in $B_{SR\,i}^{TT}$ are the values of time progress condition at state l_i in B_i. These time progress conditions are expressed on clock C^g which could be equivalently expressed on original clocks c involved in the original time progress conditions B_i. The valuation of each clock c involved in the time progress condition of B_i is computed from the valuation of $g - \rho_c$ where ρ_c is the last clock reset date of c. If the time progress condition tpc_i expressed on clock C^g allows the time step δ in TTCC component, thus, δ is also allowed by the time progress condition tpc_{l_i} expressed on original clocks of B_i. Therefore $q \rightarrow \delta r$. Executing δ has the same effect on clocks in both models, therefore $(r_{SR}^{TT}, r) \in R$.

If σ can be executed in B at state q, then from an equivalent state q_{SR}^{TT}, one can reach the state $[q_{SR}^{TT}]$(by doing β_1 actions) where the clock C^g and data of ATC components have the same values as those of q (Lemma 3).As previously, we distinguish the cases where σ is an interaction a or a delay step δ.
If σ is an intra-task interaction a, then it is not concerned by the transformation and remains intact in the obtained model. Thus we have straightforwardly $(q_{SR}^{TT}, q) \in R$ and $(r_{SR}^{TT}, r) \in R$.
If σ is an inter-task interaction a from q, then the timing constraint and guard of a are *True*. From q_{SR}^{TT} we reach $[q_{SR}^{TT}]$ by doing β_1 actions. Then we execute all possible $fail$ interactions (that are β_2 actions), to reach $[q'^{TT}_{SR}]$. At this state, if a is not conflicting, the interaction a is enabled, else the sequence $rsv_a\ ok_a$ can be executed since lemma 4 ensures that guard of ok_a is *True*. In both cases, the interaction corresponding to a brings the system in state r_{SR}^{TT}. From this state, the response corresponding to ports of a are enabled, and the next stable state $[r_{SR}^{TT}]$ is equivalent to r. Thus we have $q_{SR}^{TT} \xrightarrow{\beta_1^*} [q_{SR}^{TT}] \xrightarrow[fail]{\beta_1^*} [q'^{TT}_{SR}] \xrightarrow{a} r_{SR}^{TT} \xrightarrow{\beta_1^*} [r_{SR}^{TT}]$ and $q_{SR}^{TT} \xrightarrow{\beta_1^*} [q_{SR}^{TT}] \xrightarrow[fail]{\beta_1^*} [q'^{TT}_{SR}] \xrightarrow{rsv_a, ok_a} r_{SR}^{TT} \xrightarrow{\beta_1^*} [r_{SR}^{TT}]$. Thus $(r_{SR}^{TT}, r) \in R$.
If σ is a delay step δ then from state $[q_{SR}^{TT}]$, time can progress also by δ to reach state r'^{TT}_{SR}. At this state, place $read$ of TTCC is active and having the tpc_i sent by $[B_{SR\,i}^{TT}]$ which is the same as the time progress condition of B_i at q. From reached state r'^{TT}_{SR} we execute all possible $fail$ interactions (that are β_2 actions), to reach r_{SR}^{TT}.The time progress conditions are equal to *False*in $receive$ place in the CRP component. That is, even if some β_2 actions is possible from state r'^{TT}_{SR}, no delay is allowed at this state. Thus we have, $q_{SR}^{TT} \xrightarrow{\beta_1^*} [q_{SR}^{TT}] \xrightarrow{\delta} r'^{TT}_{SR} \xrightarrow{\beta_2} r_{SR}^{TT}$ with $(r_{SR}^{TT}, r) \in R$. ■

C. Trace inclusion between B_{SR}^{TT} and B^{TT}

We denote by B_{SR}^{TT} the initial model and by B^{TT} the resulting model of the step 2 of the transformation.

The observational equivalence cannot be proven between B_{SR}^{TT} and the resulting B^{TT}. In B^{TT} model, conflicts between internal and external interactions are resolved by local priority rules in the task. Thus when interactions a_I and a_E (respectively internal and external interactions) are possible, always the internal one will be executed. This restriction implies that the set of traces of the B^{TT} model is a subset of traces of B_{SR}^{TT} model.